CSN08x14

Scripting for Cybersecurity and Networks

Lecture 7:

Regular Expressions in Python; reading Web pages

Today's topics

- String/Text manipulation
- Regular Expressions
- External Data – Fetching Data from the Web
- Lab overview

Go to www.menti.com

Code xxx

Looking Ahead

- Class test: in your labs **6/8/9 November** (next week)
  - *50% of module*
  - *Mostly Short answer questions (a few multiple choice)*
  - *Open book*
  - *1.5 hours*
- Final Coursework: **Submit Fri 7 December**; demos **week beginning Monday 10 December**
  - *50% of module*
  - *Spec now published*
  - *Re-use code / modules from several lab exercises.*
  - *Sign up for demo slot in moodle (released nearer the time)*

String Handling

String Escape Characters

- Escapes represent special characters - to allow characters which are not easily typed on keyboard, or can't normally be embedded in strings
- Escape Char format – backslash \ followed by char you want to add to string, or char representing formatting

```
>>> print ('\'')    # ?
```

- When printed, interpreted as quote, tab, newline etc

```
>>> print ('Python is\t \'fun\'\n')
```

String Escape Characters: Examples

\\\\ backslash	*\\' single quote*
\\" double quote	*\\b backspace*
\\n newline	*\\r carriage return*
\\t tab	*\\x00 char hex value*

```
>>> '\x50'
'P'
>>> '\x50' == 'P'
True
>>> print ('\x50ython')
Python
```

Raw Strings

- *Suppress Escape Characters – sometimes don't want to use \ to escape characters – e.g. in file/dir paths*

■ **Raw String path:**

```
len('\n')   vs   len(r'\n')
```

r specifies raw string – no escaping done
(ignores escape sequences)

- ***File path:***

```
filepath = 'c:\temp\dir\file.txt'
```

Q. Why won't this work as a file Path?

- ***Escaped file path:***

```
filepath = 'c:\\temp\\dir\\file.txt'
```

■ **Raw String path:**

```
filepath = r'c:\temp\dir\file.txt'
```

r specifies raw string – no escaping done

Searching for patterns in text strings: in

`in` operator:

```
>>>'pattern' in 'string with pattern'
True
```

Pattern we are searching for

String we are searching through

Hex String Handling

We could create our own **is_hex()** function

```
def is_hex(hex_str):
    '''is_hex(str) --> bool
    returns True if the string contains only 0-9 and a-f'''
    ishex = True
    for char in hex_str:
        if not char.lower() in '0123456789abcdef':
            ishex = False
    return ishex
```

```
>>> is_hex('34vgaq7891')
False
>>> is_hex('dddab267af')
True
```

Intro to RegEx (Regular expressions)

Note: If you have never used regex before, this will be hard.
Consider working through e.g. https://regexone.com/

Regular Expressions

- Regular Expressions allow searching for/matching patterns in text

  `[Rr]ich` pattern matches `Rich or rich`

  `Rich|rich` pattern matches `Rich or rich`

- 'Regex' is the pattern to be matched against
- Language used is powerful for matching text patterns
- 'Regex' patterns can be complex due to syntax:

```
[A-Z0-9._%+-]+\@[A-Z0-9.-]+\.[A-Z]{2,4}
```

Q: what does this regex search for?

Regular expressions

- Used (almost) everywhere in computing to search for patterns in text
  - *Programming languages*
  - *Linux grep command*
  - *Windows findstr command*

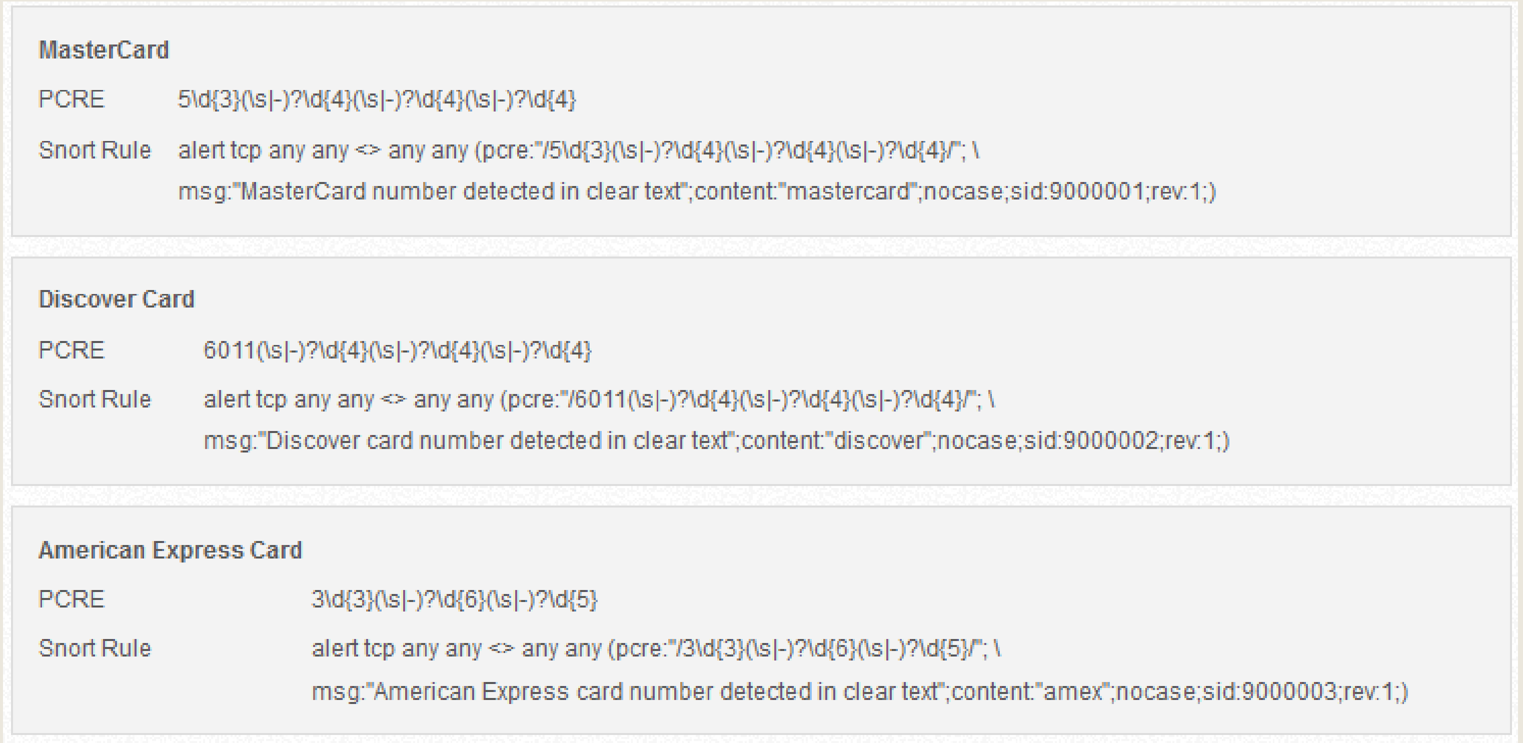
- Very versatile - can define very complex patterns
  - *Search for Email addresses? IPv4 addresses? UK postcodes?*
  - *Each application may use a slightly different "dialect"*

Some uses of Regular Expressions

- Security and networking:
  - *log file analysis – grep to SIEM systems, IDPS signatures, AV signatures*
- Forensics investigations / Incident Response
  - *searching media images for patterns such as text/emails/dates/URLs/IP data/honey tokens*

Regular Expressions in security and networking

log file analysis – grep to SIEM systems,
IDPS signatures – intrusion/attack variations,
AV signatures



MasterCard

```
PCRE        5\d{3}(\s|-)?\d{4}(\s|-)?\d{4}(\s|-)?\d{4}
Snort Rule  alert tcp any any <> any any (pcre:"/5\d{3}(\s|-)?\d{4}(\s|-)?\d{4}(\s|-)?\d{4}/"; \
            msg:"MasterCard number detected in clear text";content:"mastercard";nocase;sid:9000001;rev:1;)
```

Discover Card

```
PCRE        6011(\s|-)?\d{4}(\s|-)?\d{4}(\s|-)?\d{4}
Snort Rule  alert tcp any any <> any any (pcre:"/6011(\s|-)?\d{4}(\s|-)?\d{4}(\s|-)?\d{4}/"; \
            msg:"Discover card number detected in clear text";content:"discover";nocase;sid:9000002;rev:1;)
```

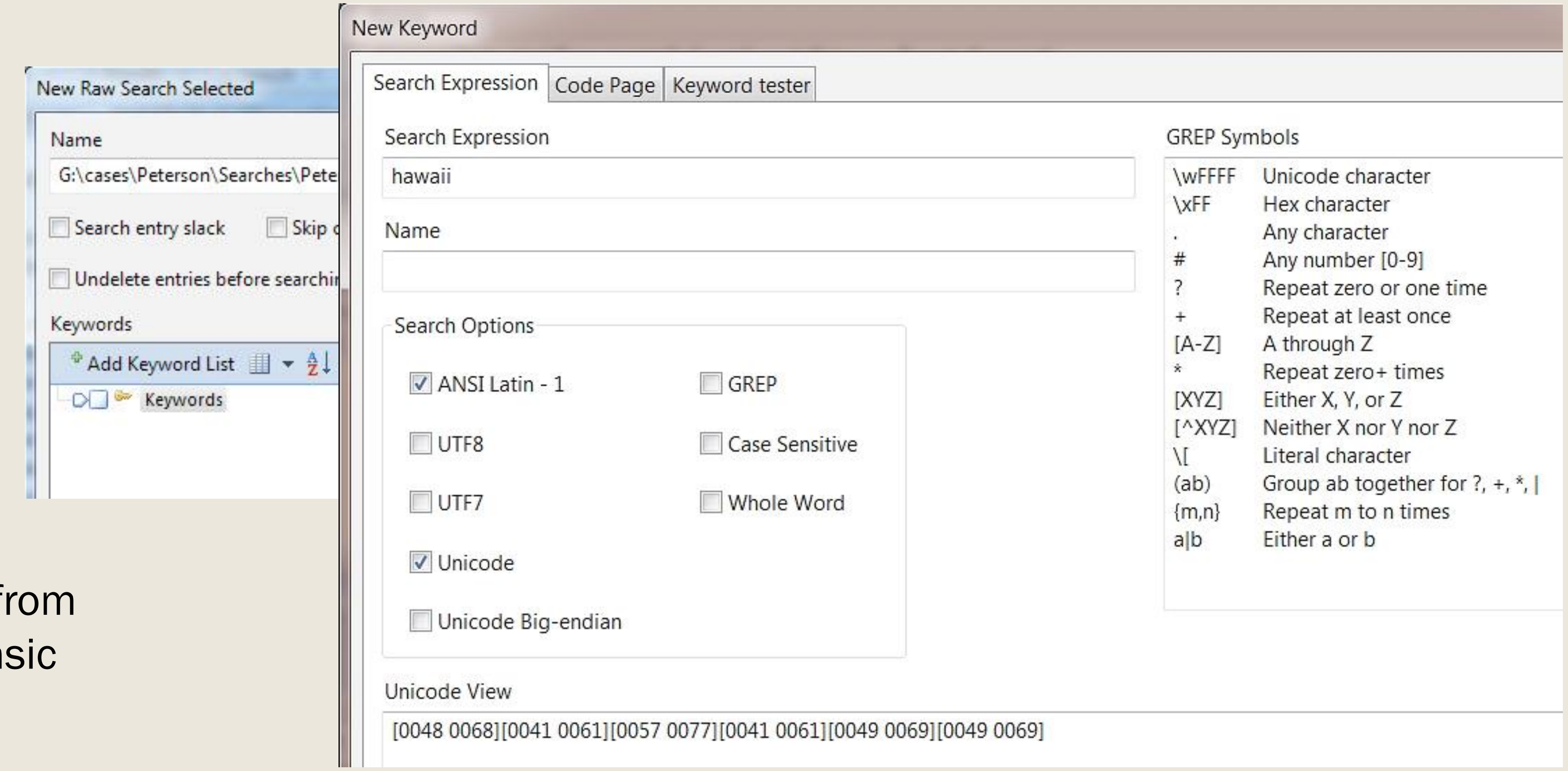
American Express Card

```
PCRE        3\d{3}(\s|-)?\d{6}(\s|-)?\d{5}
Snort Rule  alert tcp any any <> any any (pcre:"/3\d{3}(\s|-)?\d{6}(\s|-)?\d{5}/"; \
            msg:"American Express card number detected in clear text";content:"amex";nocase;sid:9000003;rev:1;)
```

See e.g. http://asecuritysite.com/forensics/snort?fname=email_cc2.pcap&rulesname=rulescc.rules

Regular Expressions in forensic investigations

Use to search media images for patterns such as investigation specific text/emails/dates/web hyperlinks – Net and Media



New Raw Search Selected

Name

G:\cases\Peterson\Searches\Pete

- [ ] Search entry slack
- [ ] Skip
- [ ] Undelete entries before searchi

Keywords

Add Keyword List

Keywords

New Keyword

Search Expression | Code Page | Keyword tester

Search Expression: hawaii

Name:

Search Options

- [✓] ANSI Latin - 1
- [ ] GREP
- [ ] UTF8
- [ ] Case Sensitive
- [ ] UTF7
- [ ] Whole Word
- [✓] Unicode
- [ ] Unicode Big-endian

GREP Symbols

\wFFFF	Unicode character
\xFF	Hex character
.	Any character
#	Any number [0-9]
?	Repeat zero or one time
+	Repeat at least once
[A-Z]	A through Z
*	Repeat zero+ times
[XYZ]	Either X, Y, or Z
[^XYZ]	Neither X nor Y nor Z
\[	Literal character
(ab)	Group ab together for ?, +, *, |
{m,n}	Repeat m to n times
a|b	Either a or b

Unicode View

```
[0048 0068][0041 0061][0057 0077][0041 0061][0049 0069][0049 0069]
```

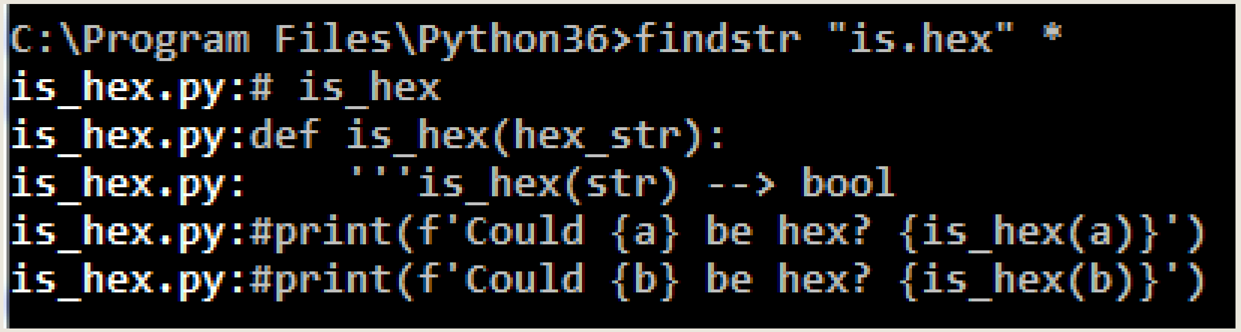
Screenshots from EnCase Forensic software

findstr – using regex on Win cmd Line

C:\Program Files\Python36 > findstr "is.hex" *

Pattern we are searching for (uses regular expressions)

Where we are searching (uses filename wildcard expansions)



```
C:\Program Files\Python36>findstr "is.hex" *
is_hex.py:# is_hex
is_hex.py:def is_hex(hex_str):
is_hex.py:    '''is_hex(str) --> bool
is_hex.py:#print(f'Could {a} be hex? {is_hex(a)}')
is_hex.py:#print(f'Could {b} be hex? {is_hex(b)}')
```

Q: Why does is.hex find underscores?

Using regex in Python

Python Regex

Python supports Regular Expressions using the **re** module (**re.search**), and **match** object

```
import re
match = re.search('pattern', 'string with pattern in')
match.group()
'pattern'
```

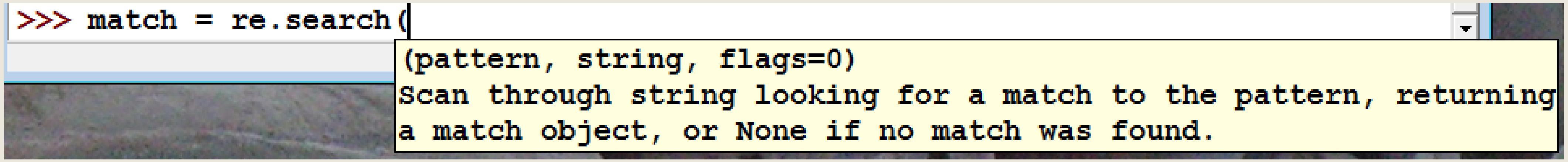
Regex Pattern we are searching for

String we are searching for the pattern

- Searches left to right
- Match object returned if pattern found
- Match object contains matching text
- group() method returns the string which matched against the pattern

Regular expressions in Python

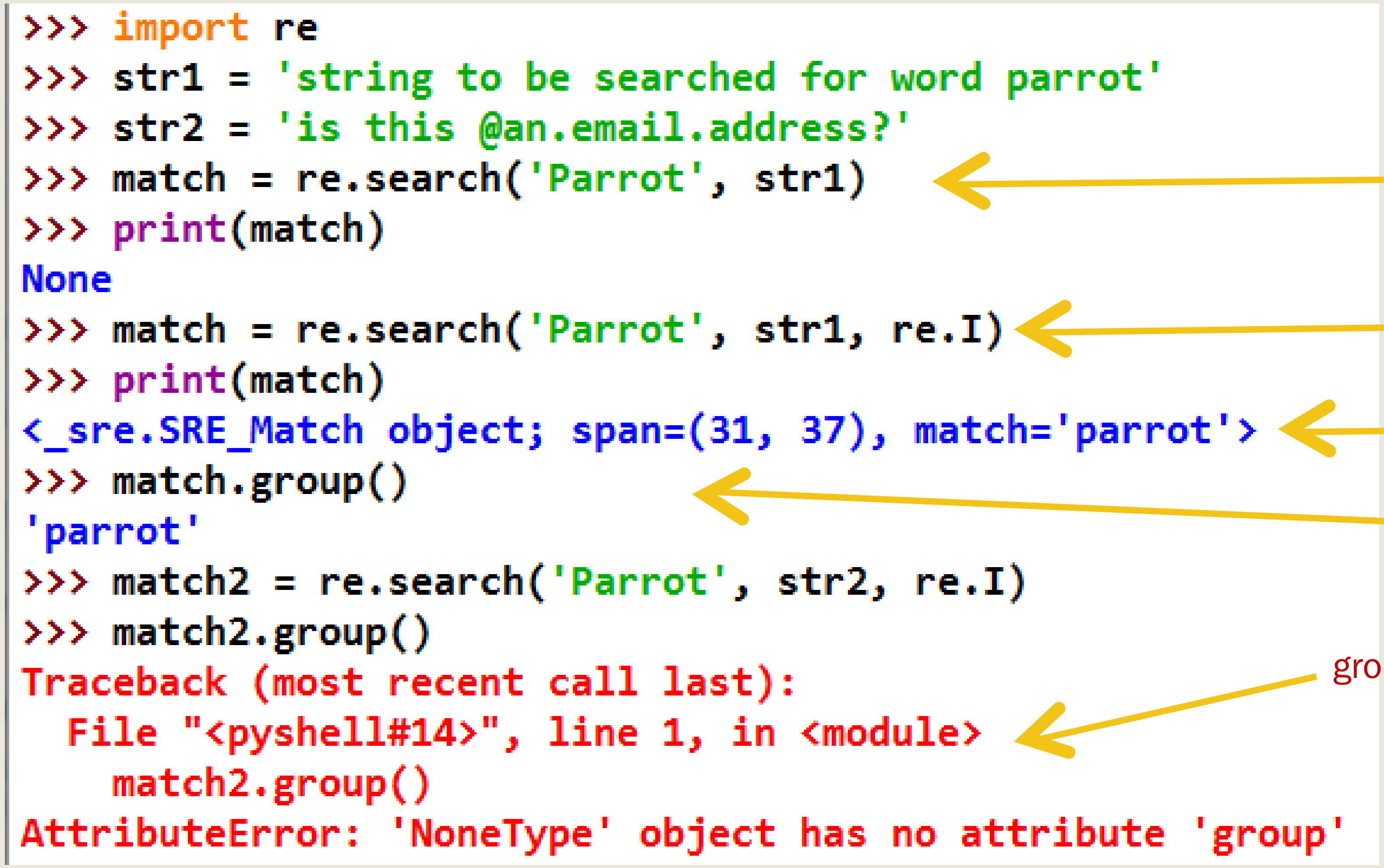
- **search()** is part of the **re** module in the standard library
- Takes a pattern, a string and optionally flag(s)
  - *Typical flag is* ***`re.I`*** *(ignorecase)*
- Returns None or a SRE_Match object
  - *Use* ***`.group()`*** *method to print contents or convert to string*



```
>>> match = re.search(
(pattern, string, flags=0)
Scan through string looking for a match to the pattern, returning
a match object, or None if no match was found.
```

- The **regex** third-party module is similar but offers better Unicode support and more functionality
- See https://docs.python.org/3.6/library/re.html

re.search() example



```
>>> import re
>>> str1 = 'string to be searched for word parrot'
>>> str2 = 'is this @an.email.address?'
>>> match = re.search('Parrot', str1)
>>> print(match)
None
>>> match = re.search('Parrot', str1, re.I)
>>> print(match)
<_sre.SRE_Match object; span=(31, 37), match='parrot'>
>>> match.group()
'parrot'
>>> match2 = re.search('Parrot', str2, re.I)
>>> match2.group()
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    match2.group()
AttributeError: 'NoneType' object has no attribute 'group'
```

None returned if no match found.
Why does this not find anything?

`re.I` makes search case insensitive

Now we get a result

Use **.group()** to display found string

group() raises exception if nothing found

Q: how can we deal with this likely Exception?

Writing regex
- expressing patterns

Building Regex patterns

- **Matching literal characters**
  - *ASCII characters match themselves*
  - ***'P'** in the regex pattern matches a **'P'** in the string being searched*
  - *Special characters which do not match themselves:* **. \ * + ? ^ $ { } [ ] | ( )**

- **Special Pattern Characters:**
  - *'.' Dot matches any character*
  - ***\w** matches any word character– same as [a-zA-Z0-9_]*
  - ***\d** matches numeric digits – same as [0-9]*
  - ***\t, \n** tab, newline*
  - ***\s** whitespace - same as [\t\n\r\f\v]*

- **Matching Special Characters:**
  - ***\.** Matches '.'*
  - ***\\** Matches '\'*
  - *Known as an Escape Sequence -* \ *used to escape special character*

Examples 1

'.'   matches any single character

```
match = re.search('pat...n', 'string with pattern in')
# check if pattern found
if match:
      print (match.group())

'pattern'
```

- Raw strings typically used for regex pattern as no escaping performed

Q. Will this match 'string with patt@rn in' ?

Examples 2

more specific than '.':

- `\w` **matches any single word character (letters, digits, underscore)**
  - *Same as* [a-zA-Z0-9_]

```
match = re.search('pat\w\w\w\w', 'string with pattern in')
if match: print (match.group())
'pattern'
```

Q. Will this regex match 'string with patt@rn in' ?

Examples 3

- `\d` **matches any digit numeric character [0-9]**
- `\s` **matches any whitespace character (Space,Tab,Newline)**

```
match = re.search('pin:\s\d\d\d\d','pin: 1223 pattern')
if match: print (match.group())
     'pin: 1223'
```

Q. Will regex pattern match 'stuff pin:\t6666 stuff' ?
Q. Will regex pattern match r'stuff pin:\t6666 stuff' ?

Be careful with case!

- Patterns themselves are case sensitive,
- Upper case versions of character specs make search Negative!:

`\d`	matches any digit numeric character [0-9]
`\D`	matches anything **except** digit numeric character [0-9]
`\w`	matches any single word character [a-zA-Z0-9_]
`\W`	matches anything **except** single word character [a-zA-Z0-9_]
`\s`	matches any whitespace character (Space,Tab,Newline)
`\S`	matches anything **except** a whitespace character

Escape Sequences

- **\ escapes the character after - removing the specialness!**
- **Special Characters:** `. \ * + ? ^ $ { } [ ]  | ( )`

- `\.` **matches a single dot '.' char**

```
match = re.search('192\.168\.\d\.\d','private ip 192.168.5.3')
if match: print (match.group())

    '192.168.5.3'
```

- **If not sure, you can escape any character** (e.g. if you're not sure about @ use \@ instead)

Repeating Pattern Characters (quantifiers)

- **Special characters to specify (quantify) how many chars of a type to match**

  *+ **One or more** of the specified characters*

  ** **Zero or more** of the specified characters*

  *? **Zero or one** of the preceding character or group*

  ***{n} exactly n** of the specified chars/patterns*

  ***{n,m} between n and m** of the specified chars/patterns*

Example

- `s+` matches one or more literal 's' characters

```
match = re.search('s+', 'python talk: ssssssssssss ssssss')

if match: print (match.group())
ssssssssssss
```

Q. Will regex pattern r's{5} ' match anything? What?

Quantifiers examples

- `\d+` matches one or more digit characters [0-9]

```
match = re.search('\d+\.\d+\.\d+\.\d+', 'stuff 146.176.10.2 stuff')

if match: print (match.group())
146.176.10.2


match = re.search('\d+\.\d+\.\d+\.\d+', 'stuff 1468.176.10.2 stuff')

if match: print (match.group())
1468.176.10.2
```

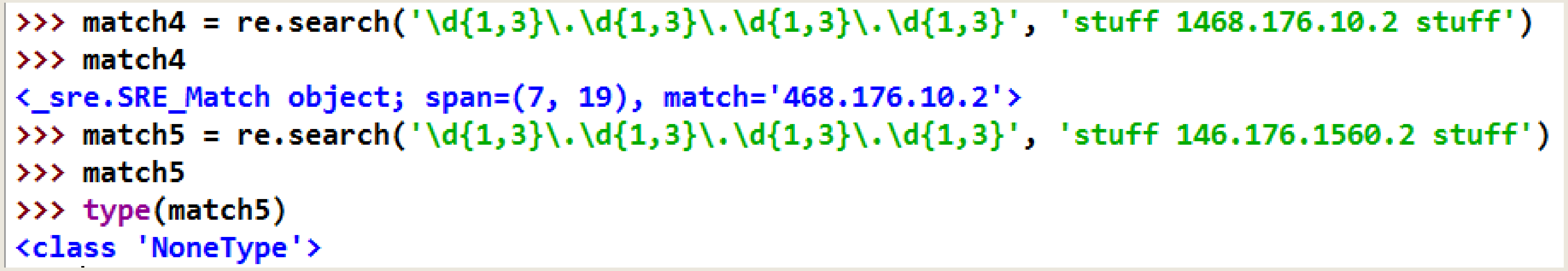
Q. Did you expect this? How could we prevent it?

- *** and + will match as far as possible** – known as greedy operators

Q. What will regex pattern 's.*' match?

Quantifiers examples ctd

- `\d{1,3}` matches between 1 and 3 digit characters [0-9]



```
>>> match4 = re.search('\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}', 'stuff 1468.176.10.2 stuff')
>>> match4
<_sre.SRE_Match object; span=(7, 19), match='468.176.10.2'>
>>> match5 = re.search('\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}', 'stuff 146.176.1560.2 stuff')
>>> match5
>>> type(match5)
<class 'NoneType'>
```

So this works better, but
- How can we ensure the first part is also only 3 digits? → anchors, or match whitespace too
- Valid IP addresses can only be up to 255 in each part - how could we search for that?
  → complex regex, or parse results and reject if any one of the parts is not valid

[ ] : Match a single char from a set

Square Brackets [ ]

[abc] set of characters - match ANY SINGLE character in the set
- equivalent to [a OR b OR c]

```
match = re.search('\d+[- ]\d+ d+', '0131 334 7777')
if match: print (match.group())
0131 334 7777

match = re.search('\d+[- ]\d+', '0131-334 7777')
if match: print (match.group())
0131-334
```

[] contains set of chars

Matches on
[– or SPACE]
between digits

Repetition with Sets

- ***Sets of characters to match on using [ ]***
  - **Repeat any char in set with * or + , make optional with ?**
    - *+ One or more of the specified characters*
    - ** Zero or more of the specified characters*
    - *? Zero or one of the specified chars*

```
match = re.search('[\d.]+', 'stuff stuff 146.176.165.12 stuff')

if match: print (match.group())
146.176.165.12

match = re.search('[\w.-]+@[\w.-]+', 'stuff r.macf@napier.ac.uk stuff')

if match: print (match.group())
r.macf@napier.ac.uk
```

Note that these regex aren't fantastically specific – e.g. the first will also match 4 consecutive dots!

Also note inside a square bracket we don't need to escape the dot

Repetition with Sets

- ***Sets of characters to match on using [ ]***
  - **Repeat any char in set with * or +**
    - *+ One or more of the specified characters*
    - ** Zero or more of the specified characters*

```
>>> a = 'stufff 222222222'
>>> b = 'stuffff 0131 - 222890018'
>>> c = 'tel: 0131 22289962'
```

Q. Which of the regex below will correctly find the entire phone numbers in all 3 test strings?

```
>>> reg1='\d+[ -]+\d+'
>>> reg2='\d*[ -]?\d+'
>>> reg3='\d*[ -]*\d+'
>>> reg4='\d+[ -]*\d+'
```

( ): Regex Groups

1. *Use round brackets to create a group*
2. *Groups can be used to scope other operators (creates a sub-pattern within the brackets)*
3. *Groups can be used to extract sub-patterns*
4. *Advanced: Use to match on same group later in main pattern!*

```
match = re.search('(\w+)@(\w+.com)','e: jc@montypython.com  stuff')

print (match.group())
jc@montypython.com
print (match.group(1))
jc
print (match.group(2))
montypython.com

print (match.groups())
('jc','montypython.com')
```

Round brackets are used to define groups

Group method takes optional argument to select which group to return
Good way to find a pattern and extract a sub-pattern
() or (0) matches everything

Group**s** method returns a tuple with all the sub groups as elements

Regex Groups: demo

- Extract area code and phone number as separate sub-patterns

```
>>> import re
>>> m=re.search('(\d+)[- ]*(\d+)', 'stufff 0131-22222222')
>>> m.group()
'0131-22222222'
>>> m.groups()
('0131', '22222222')
>>> m.group(1)
'0131'
>>> m.group(2)
'22222222'
>>> if m: print (m.groups())
('0131', '22222222')
>>> m2=re.search('(0131|0141)[- ]*(\d+)', 'stufff 0131-22222222')
```

| can be used for OR

Find all matches: re.findall()

- re.search() only finds/returns first match
- **re.findall()** method finds all matches of a pattern,
  and returns the matches as a list

```
s1 = 'stuff stuff 146.176.122.12 stuff 146.176.123.88 stuff stuff'

matches = re.findall('\d+.\d+.\d+.\d+', s1)

print (matches)
['146.176.122.12', '146.176.123.88']
```

re.findall() method to search through files

```
f = open(filename, 'r')
match_list = re.findall('[\d+.]+', f.read())

print (match_list)
['146.176.165.12', '146.176.165.23','146.176.165.1', '146.176.165.7']



f2 = re.findall('def\s\w+',open(r'dict_crack_with_args.py').read())

print (f2)
['def dict_attack', 'def is_hex', 'def main']
```

Could open and read file within the findall() – but harder to understand and troubleshoot

re.findall() method with Groups

```
S2 = 'stuff jc@montypython.com stuff stuff rich@gmail.com'
matches = re.findall('(\w+)@(\w+.com)', s2)

print (matches)
[('jc', 'montypython.com'), ('rich', 'gmail.com')]
print (matches[0])
('jc', 'montypython.com')
```

Returns a list of tuples

```
for match in matches:
   user, domain = match
   print (user, '-', domain)

jc - montypython.com
rich - gmail.com
```

Unpack each tuple in list
into individual variables

greedy and minimal matching

- Greedy matching means that the RegEx will match "as much as possible"
- Add ? after the qualifier to make the search non-greedy or minimal, i.e. match as little as possible

```
>>> html1 = '<head>Title</head><body>text</body>'
>>> re.findall('<.+>', html1)
['<head>Title</head><body>text</body>']
>>> re.findall('<.+?>', html1)
['<head>', '</head>', '<body>', '</body>']
```

Regex Patterns: Anchoring

- ***Anchors*** *- match positions in search string – not any char*

  ^ start of string

  $ end of string

```
str = 'Rich is not rich'

matches = re.findall('^[Rr]ich', str)
print (matches)
['Rich']
```

- Only matches on first 'Rich' not second, due to the ^
- Useful if position of a search pattern is at beginning of a line

```
matches = re.findall('[Rr]ich$', str)
print (matches)
['rich']
```

Regex Patterns: Anchoring

Use both ^ and $ to match entire string

```
>>> str = 'Rich'
>>> match = re.search('^[Rr]ich$', str)
>>> if match: print (match.group())
Rich

>>> str = '@Rich'
>>> match = re.search('^[Rr]ich$', str)
>>> if match: print (match.group())
…    else: print ('Not Found')
Not Found
```

Only matches pattern if entire string matches exactly

- Can be used for pattern-based argument validation
- Parsing text – one token/word at a time

Regex Anchors example

```
rule = """permitt tcp 192.168.10.0 0.0.0.255
         host 192.168.20.5 eq www"""
```

firewall rule we want to Parse and validate
'permitt' is invalid command

```
tokens = rule.split()
```

Splits rule string into parts (on spaces)

Checks whether action is permit or deny with nothing before or after

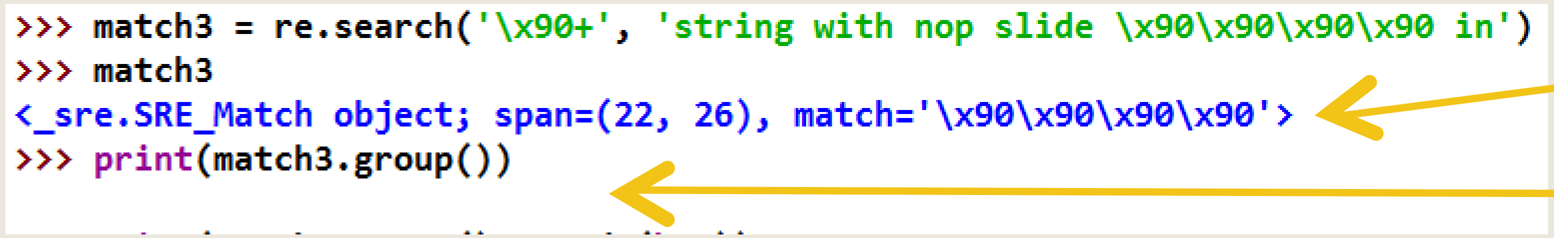
```
if not re.search('^(permit|deny)$',tokens[0]):
    print ('Invalid syntax: '+ tokens[0])
Invalid syntax: permitt
```

Matching hex in text

- **`\x90+`** matches one or more hex 90 characters

```
match = re.search('\x90+', 'string with nop slide \x90\x90\x90\x90 in')
```

- But this isn't as easy to manage as a "proper" string being returned:



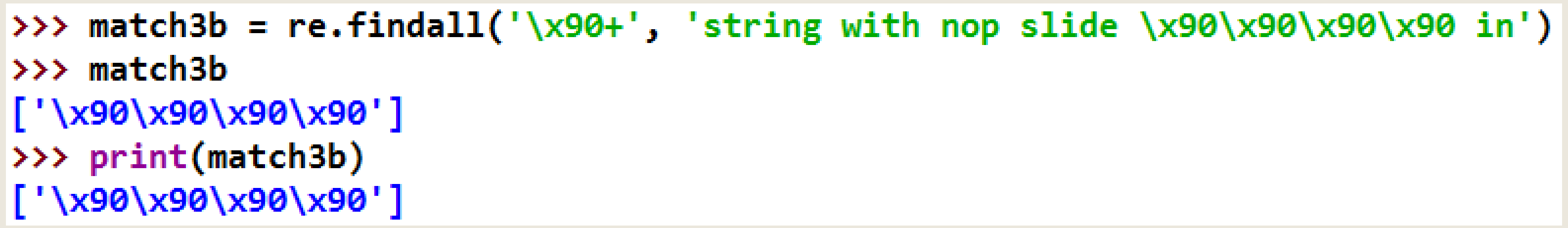
```
>>> match3 = re.search('\x90+', 'string with nop slide \x90\x90\x90\x90 in')
>>> match3
<_sre.SRE_Match object; span=(22, 26), match='\x90\x90\x90\x90'>
>>> print(match3.group())

```

Clearly returns the match as expected

But the print statement returns a blank line

- Use re.findall instead, works better here



```
>>> match3b = re.findall('\x90+', 'string with nop slide \x90\x90\x90\x90 in')
>>> match3b
['\x90\x90\x90\x90']
>>> print(match3b)
['\x90\x90\x90\x90']
```

Developing RegEx

Regular Expression Patterns can become very complex and code is dense

```
\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}\b
```

Time consuming to create and debug

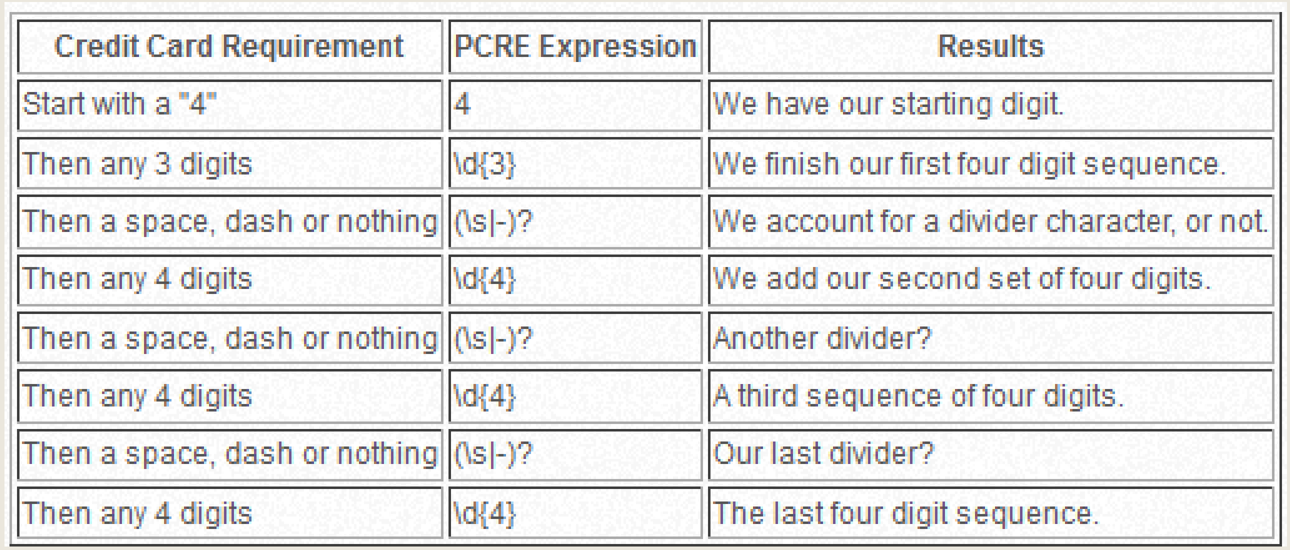
- Experiment in the Interpreter
- Use an online tester e.g. https://pythex.org/
- Set up a test string and print out what pattern matches, using a small test search string
- Create patterns in small sections, make pattern match something, then expand pattern.

```
>>> str = 'stuff stuff 146.176.165.12 stuff'
>>> print (re.findall('[\d+.]+', str))
['146.176.165.12']
```

- When you think you have the right regex test things it shouldn't match!

Example: Snort rule

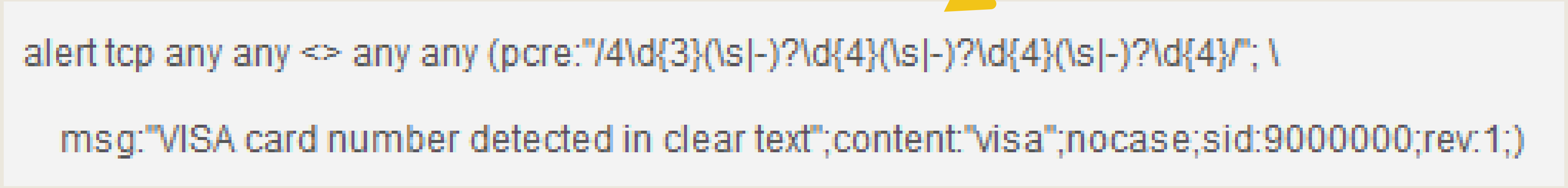
- Snort Rules for clear text Credit Card Detection



Credit Card Requirement	PCRE Expression	Results
Start with a "4"	4	We have our starting digit.
Then any 3 digits	\d{3}	We finish our first four digit sequence.
Then a space, dash or nothing	(\s|-)?	We account for a divider character, or not.
Then any 4 digits	\d{4}	We add our second set of four digits.
Then a space, dash or nothing	(\s|-)?	Another divider?
Then any 4 digits	\d{4}	A third sequence of four digits.
Then a space, dash or nothing	(\s|-)?	Our last divider?
Then any 4 digits	\d{4}	The last four digit sequence.

Regex for Visa Credit Card

"Quite a simple example"



```
alert tcp any any <> any any (pcre:"/4\d{3}(\s|-)?\d{4}(\s|-)?\d{4}(\s|-)?\d{4}/"; \
  msg:"VISA card number detected in clear text";content:"visa";nocase;sid:9000000;rev:1;)
```

Help & testing

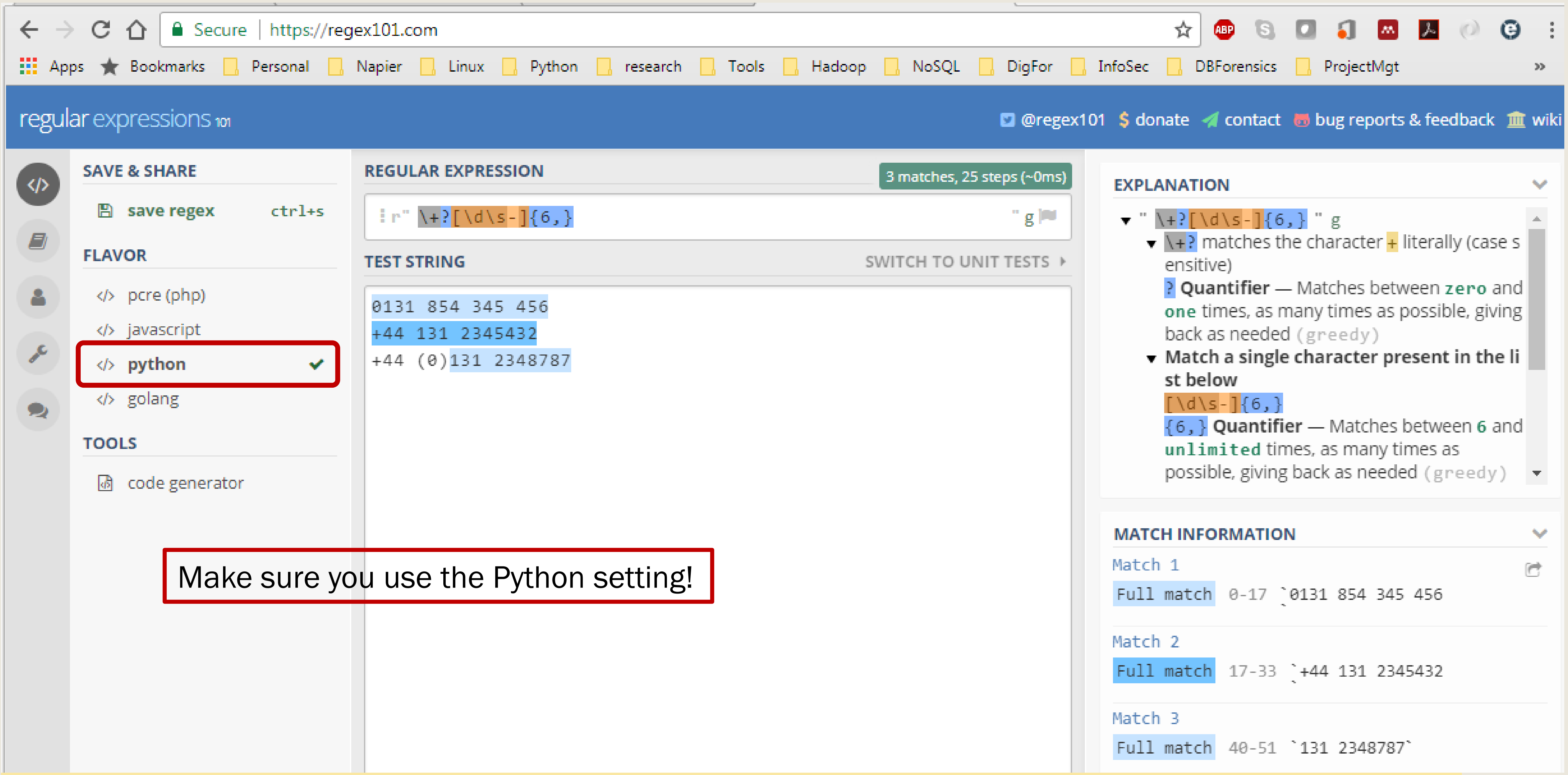
- For an interactive regex tutorial, that starts right at the beginning, go to https://regexone.com/.
- Interactive tester for Python RegEx: http://pythex.org/
- Interactive tester with a Python flavour that also explains each regex element http://regex101.com.

- https://www.regular-expressions.info/tutorial.html
- Python docs: https://docs.python.org/3/library/re.html

- https://www.youtube.com/watch?v=zN8rwVXwRUE (Python Regular Expressions Tutorial (45 mins))

- If you google for regex, make sure you fully understand and test any solutions you find, and of course acknowledge the source in your code!

http://regex101.com



REGULAR EXPRESSION 3 matches, 25 steps (~0ms)

```
r" \+?[\d\s-]{6,} " g
```

TEST STRING

```
0131 854 345 456
+44 131 2345432
+44 (0)131 2348787
```

EXPLANATION

" \+?[\d\s-]{6,} " g
- +? matches the character + literally (case sensitive)
  ? Quantifier — Matches between zero and one times, as many times as possible, giving back as needed (greedy)
- Match a single character present in the list below
  [\d\s-]{6,}
  {6,} Quantifier — Matches between 6 and unlimited times, as many times as possible, giving back as needed (greedy)

MATCH INFORMATION

```
Match 1
Full match  0-17   `0131 854 345 456
Match 2
Full match  17-33  `+44 131 2345432
Match 3
Full match  40-51  `131 2348787`
```

Make sure you use the Python setting!

Getting information from web pages

Using the urllib library

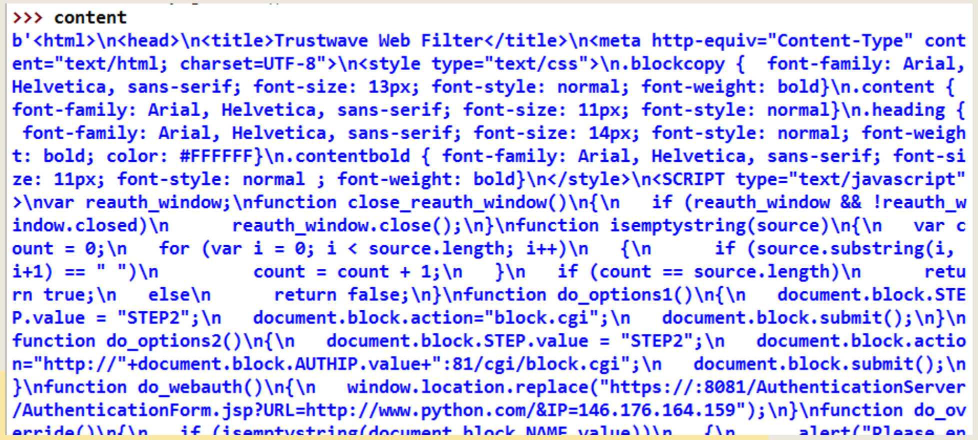
Python Data from Web: urllib

- Urllib.request library – Get data from the Web
- Use urllib.request.urlopen() to return a web page object from a URL (similar to a local file object)

```
import urllib.request
url = "http://www.python.org"
webpage = urllib.request.urlopen(url)
```

- Read text from web page object as normal

```
content = webpage.read()
```



```
>>> content
b'<html>\n<head>\n<title>Trustwave Web Filter</title>\n<meta http-equiv="Content-Type" cont
ent="text/html; charset=UTF-8">\n<style type="text/css">\n.blockcopy {  font-family: Arial,
Helvetica, sans-serif; font-size: 13px; font-style: normal; font-weight: bold}\n.content {
font-family: Arial, Helvetica, sans-serif; font-size: 11px; font-style: normal}\n.heading {
 font-family: Arial, Helvetica, sans-serif; font-size: 14px; font-style: normal; font-weigh
t: bold; color: #FFFFFF}\n.contentbold { font-family: Arial, Helvetica, sans-serif; font-si
ze: 11px; font-style: normal ; font-weight: bold}\n</style>\n<SCRIPT type="text/javascript"
>\nvar reauth_window;\nfunction close_reauth_window()\n{\n   if (reauth_window && !reauth_w
indow.closed)\n      reauth_window.close();\n}\nfunction isemptystring(source)\n{\n   var c
ount = 0;\n   for (var i = 0; i < source.length; i++)\n   {\n      if (source.substring(i,
i+1) == " ")\n         count = count + 1;\n   }\n   if (count == source.length)\n      retu
rn true;\n   else\n      return false;\n}\nfunction do_options1()\n{\n   document.block.STE
P.value = "STEP2";\n   document.block.action="block.cgi";\n   document.block.submit();\n}\n
function do_options2()\n{\n   document.block.STEP.value = "STEP2";\n   document.block.actio
n="http://"+document.block.AUTHIP.value+":81/cgi/block.cgi";\n   document.block.submit();\n
}\nfunction do_webauth()\n{\n   window.location.replace("https://:8081/AuthenticationServer
/AuthenticationForm.jsp?URL=http://www.python.com/&IP=146.176.164.159");\n}\nfunction do_ov
erride()\n{\n   if (isemptystring(document.block.NAME.value))\n   {\n      alert("Please en
```

Python Data from Web: exception handling

- **Exception Handling if urlopen fails**

```
try:
    url = 'http://www.nonexistantpage.com'
    webpage = urllib.request.urlopen(url)
except:
    print ('error loading web page:', url)
```

- **Check HTTP Status Code** (see https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)

```
url = 'http://www.python.com'
webpage = urllib.request.urlopen(url)
webpage.getcode()
200
```

Practical Lab 07

- RegEx
- Reading web pages (webpage_get.py)
- email_analysis.py

Script to get Webpage Contents from a URL

`webpage_get.py`

```
# Script:   webpage_get.py
# Desc:     Fetches data from a webpage.
# Author:   PL & RM
# Modified:  Oct 2017
#
import sys, urllib.request

def wget(url):
    ''' Retrieve a webpage via its url, and return its contents'''
    print ('[*] wget()')
    # open url like a file, based on url instead of filename
    webpage = None # ADD YOUR CODE TO OPEN URL HERE
    # get webpage contents
    page_contents = None # ADD YOUR CODE HERE
    return page_contents

def main():
    # set test url argument
    sys.argv.append('http://www.napier.ac.uk/Pages/home.aspx')

    # Check args
    if len(sys.argv) != 2:
        print ('[-] Usage: webpage_get URL')
        return

    # Get web page
    print (wget(sys.argv[1]))

if __name__ == '__main__':
        main()
```

Open and read the webpage contents, then return

Test URL argument

Get webpage content and print

Q. What do we need to add to the wget() code?

Trying to open a webpage…our wget() function

- Python now by default checks for a valid ssl certificate of a webpage before opening it (to prevent e.g. man-in-the-middle attacks)
- This may result in the error:
  urllib.error.URLError: <urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:748)>
- To fix the error:

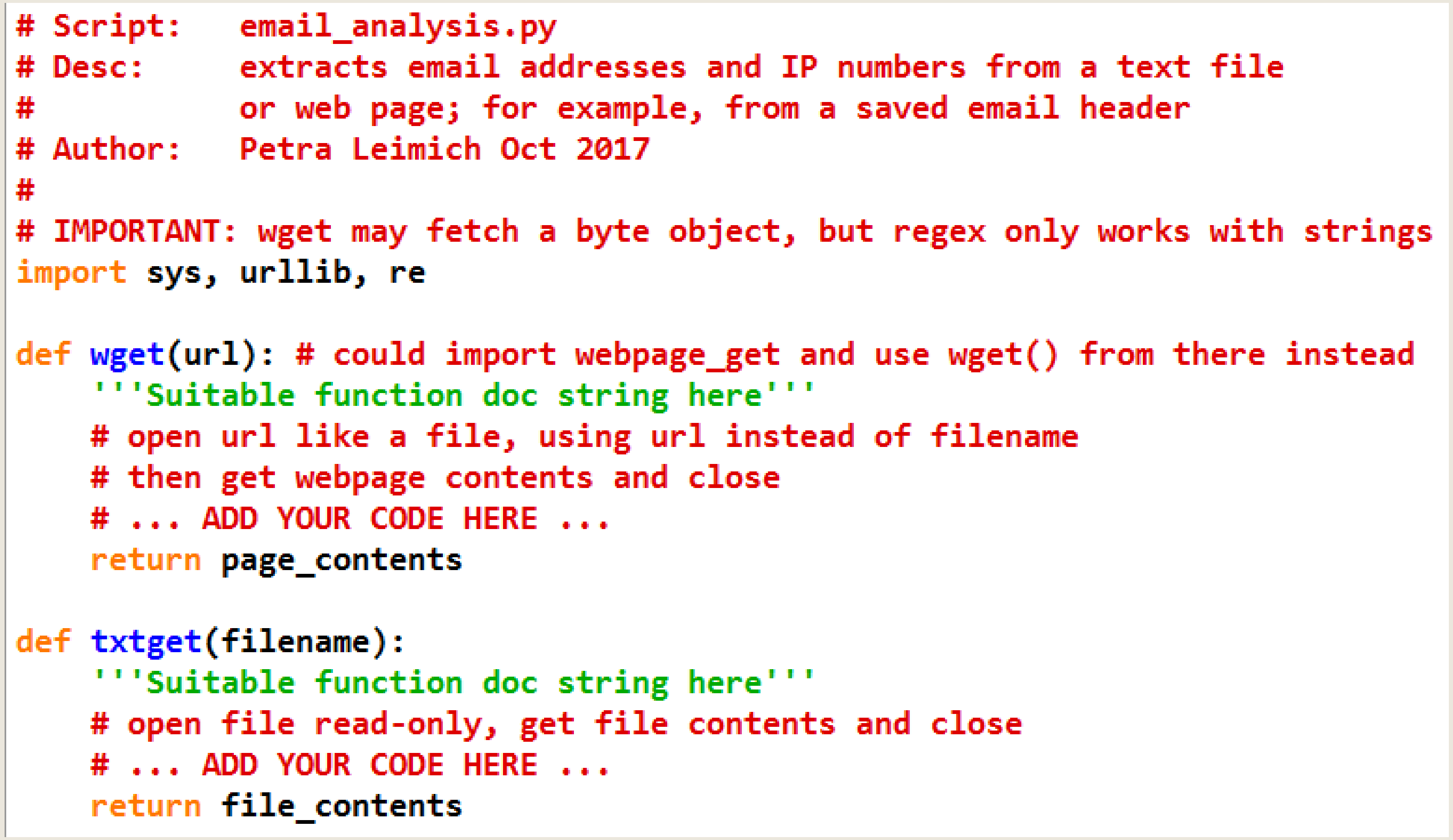
You should only do this if you trust the website

1. **`import ssl`**
2. *in wget(), add the following line at the top:*
   `context = ssl._create_unverified_context()`
3. *Change the urlopen call from* *webpage = urllib.request.urlopen(url)*
   to
   `webpage = urllib.request.urlopen(url, context=context)`

Putting it together: email header analysis

- Email headers contain information about the route the email has taken from sender to recipient
- Valuable for identifying malicious emails
- E.g. http://asecuritysite.com/email02.txt
- Look at e.g. IP addresses and email addresses

Script for extracting emails and IP addresses part 1



```python
# Script:   email_analysis.py
# Desc:     extracts email addresses and IP numbers from a text file
#           or web page; for example, from a saved email header
# Author:   Petra Leimich Oct 2017
#
# IMPORTANT: wget may fetch a byte object, but regex only works with strings
import sys, urllib, re

def wget(url): # could import webpage_get and use wget() from there instead
    '''Suitable function doc string here'''
    # open url like a file, using url instead of filename
    # then get webpage contents and close
    # ... ADD YOUR CODE HERE ...
    return page_contents

def txtget(filename):
    '''Suitable function doc string here'''
    # open file read-only, get file contents and close
    # ... ADD YOUR CODE HERE ...
    return file_contents
```

What do we need to add?

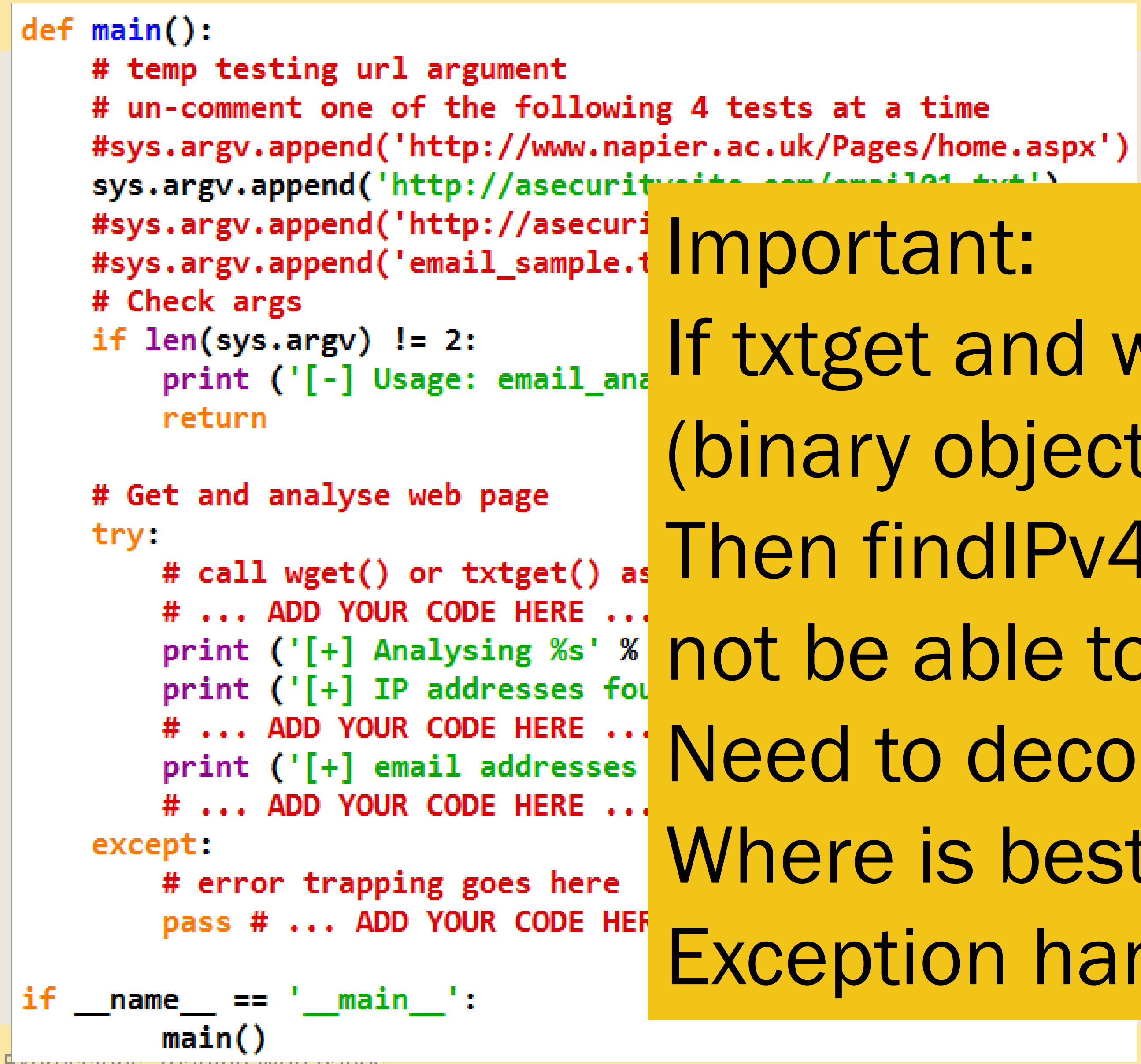
Script for extracting emails and IP addresses part 2

```python
def findIPv4(text):
    '''Suitable function doc string here'''
    ips = [] # ... ADD YOUR CODE HERE ...
    return ips

def findemail(text):
    '''Suitable function doc string here'''
    emails = [] # ... ADD YOUR CODE HERE ...
    return emails
```

What do we need to add?
Use re.search() or re.findall()?

Script for extracting emails and IP addresses part 3



```
def main():
    # temp testing url argument
    # un-comment one of the following 4 tests at a time
    #sys.argv.append('http://www.napier.ac.uk/Pages/home.aspx')
    sys.argv.append('http://asecurit
    #sys.argv.append('http://asecuri
    #sys.argv.append('email_sample.t
    # Check args
    if len(sys.argv) != 2:
        print ('[-] Usage: email_ana
        return

    # Get and analyse web page
    try:
        # call wget() or txtget() as
        # ... ADD YOUR CODE HERE ...
        print ('[+] Analysing %s' %
        print ('[+] IP addresses fou
        # ... ADD YOUR CODE HERE ...
        print ('[+] email addresses
        # ... ADD YOUR CODE HERE ...
    except:
        # error trapping goes here
        pass # ... ADD YOUR CODE HER

if __name__ == '__main__':
    main()
```

What do we need to

Important:
If txtget and wget return bytes (binary object)
Then findIPv4 and findemail will not be able to find anything
Need to decode at some point.
Where is best?
Exception handling?

Script for extracting emails and IP addresses Enhancements part 1

- Although opening and reading a local file or a web page is quite similar, we need to apply the appropriate method!!

- Could do this manually via main()

Better:

- Auto-detect whether the argument is a URL or local file, and automatically call the appropriate function to open and read the object
- How?

Script for extracting emails and IP addresses Enhancements part 2

- IP addresses must have a number between 0-255 in each quartet (0.0.0.0-255.255.255.255)
- Our regex might also find e.g. 999.6.123.12
- How to filter out?

Script for extracting emails and IP addresses Enhancements part 3

- Email headers typically contain multiple occurrences of email addresses. Ideally, your script should count the number of occurrences and print each email address only once with its count, instead of repeating email addresses found more than once.

- How?